Parte a.- Programe la función *integral* en el archivo *parte-a.c*. Esta función calcula numéricamente la integral de una función que se recibe como parámetro. Pruebe su solución con el archivo *test-parte-a.c*. El encabezado de *integral* es:

```
typedef double (*Funcion)(void *ptr, double x);
double integral(Funcion f, void *ptr,
                double xi, double xf, int n);
```

Ud. debe calcular numéricamente $\int_{xi}^{xf} f(ptr, x)\,dx$ usando el método de los trapecios, en donde *n* es el número de trapecios usados para aproximar el área bajo la curva. Para ello use la siguiente fórmula:

$$\int_{xi}^{xf} f(ptr, x)\,dx \approx h\cdot\left[\frac{f(ptr, xi)+f(ptr, xf)}{2}+\sum_{k=1}^{n-1} f(ptr, xi+k\cdot h)\right]$$

con $h=\frac{xf-xi}{n}$. El puntero *ptr* se usa para pasar parámetros adicionales a la función en caso de necesidad. A modo de ejemplo suponga que Ud. dispone de la función *g(x, y)*. En el código de más abajo la función *integral_g_dx* usa *integral* para calcular numéricamente $\int_{xi}^{xf} g(x, y)\,dx$ en donde *y* es una constante. Observe que como *g* no posee el tipo requerido por *integral*, se introduce *g_aux* que sí posee el tipo requerido.

```
double g_aux(void *ptr, double x) {
  double y= *(double*)ptr;
  return g(x, y);
}

double integral_g_dx(double xi, double xf, double y, int n) {
  return integral(g_aux, &y, xi, xf, n);
}
```

Parte b.- Programe la función *integral_par* en el archivo *parte-b.c*. Esta función calcula la misma integral de la parte b pero en paralelo usando *p* cores recurriendo a *p*n* trapecios. Para ello use *fork* para crear *p* procesos pesados. Utilice un pipe por cada proceso hijo para que este le entregue al padre el área del subintervalo de *x* calculado. Pruebe su solución con el archivo *test-parte-b.c*. El encabezado de la función es:

```
double integral(Funcion f, void *ptr,
                double xi, double xf, int n, int p);
```

Parte c.- Esta parte consiste en paralelizar el cálculo numérico de $\int_0^1 f(x)\,dx$ usando sockets. Para ello se descompone el intervalo [0,1] en 1000 subintervalos de la forma $[i*\Delta x, (i+1)*\Delta x]$ con $\Delta x=\frac{1}{1000}$ e *i* tomando valores entre 0 y 999. Para esta paralelización se utiliza un número desconocido de computadores single-core conectados en red y que actuarán como clientes.

Un proceso servidor (comando *./serv*) y se encarga de enviar subintervalos a los clientes y mostrar el resultado final. El servidor acepta conexiones de los clientes a través del puerto 3000 y crea para cada uno de ellos un thread que se encarga de enviar un subintervalo a ese cliente, esperar la recepción de la integral parcial, enviar de inmedianto un nuevo subintervalo, y así hasta que se acaben los 1000 subintervalos. Entonces muestra el resultado final como la suma de las integrales parciales.

Cada cliente (comando *./cliente*) calcula numéricamente la integral parcial de *f* en cada subintervalo *[xi, xf]* recibido del servidor. Esto lo hace llamando a la función *double integral_f(double xi, double xf)* dada en el archivo *fun.c*. Esto toma tiempo porque requiere evaluar *f* en muchísimos puntos. Después envía el resultado al servidor, recibe de inmediato un nuevo subintervalo (si aún quedan) y continúa calculando, sin permanecer ocioso en ningún momento.

El siguiente es un ejemplo de uso que muestra el servidor trabajando con 3 clientes. Los clientes pueden llegar en cualquier momento. El despliegue de los comandos se muestra en orden cronológico.

servidor	*cliente 1*	*cliente 2*	*cliente 3*
`$ ./serv`			
	`$ ./cliente`		
`env [0.000, 0.001]`	`rec [0.000, 0.001]`	`% ./cliente`	
`env [0.001, 0.002]`		`rec [0.001, 0.002]`	
`env [0.002, 0.003]`	`rec [0.002, 0.003]`		`% ./cliente`
`env [0.003, 0.004]`			`rec [0.003, 0.004]`
`env [0.004, 0.005]`		`rec [0.004, 0.005]`	
`env [0.005, 0.006]`			`rec [0.005, 0.006]`
`env [0.006, 0.007]`	`rec [0.006, 0.007]`		
…	…	…	…
`env [0.999, 1.000]`		`rec [0.999, 1.000]`	
`integral= 0.323402`	`$`	`$`	`$`

Programe el servidor en el archivo *serv.c* y el cliente en *cliente.c* de manera que reproduzcan exactamente las salidas estándares mostradas en este ejemplo de uso. No se preocupe por el término del servidor (termina con *control-C*). Sí debe preocuparse por el término de los clientes.

Recursos

Baje *t4.zip* de U-cursos y descomprímalo. El directorio *T4* contiene el *Makefile* que le ayudará a compilar su tarea, *integral.h* (con los encabezados de las funciones pedidas), *libjsocket.c* (para manejo de sockets), *util.c* (función *leer*, *sendstr*, etc.), *fun.c* (la función *integral_f*), *test-parte-a.c* (el test de prueba para la parte a) y *test-parte-b.c* (el test de prueba para la parte b). El archivo *Makefile* contiene instrucciones adicionales para la compilación y ejecución.

Entrega

Ud. debe entregar por medio de U-cursos un archivo *.zip* con los archivos *parte-a.c*, *parte-b.c*, *serv.c* y *cliente.c*. ¡No incluya archivos binarios! Si alguna de las partes no funciona, la nota es 1.0. Se descontará medio punto por día de atraso. No se consideran los días sábado, domingo y festivos.